SEVENSTAX Ajax User Manual

Revision No.:	1.4
State:	Release
Author:	SEVENSTAX GmbH

Initial version:	08/07/09
Last change:	19/02/10
Changed by:	Ralf Schwarzer
Last Review:	17/02/2010
Publication:	1.4
Filename:	SEVENSTAX_Ajax_UserManual_V01_04.odt

Copyright (c) 2010 by SEVENSTAX GmbH
This document is an intellectual property of SEVENSTAX GmbH. Unauthorized copying and distribution is prohibited.

Table of Contents

1 Abstract....4

2 Product Definition....5

2.1 Features....5

2.2 Requirements....6

2.3 Ajax Modules....6

2.4 How it works....7

2.4.1 Send HTTP Request from Browser ⇨ Device....7

2.4.2 Process HTTP Request on Device....7

2.4.3 Send HTTP Response from Device ⇨ Browser....7

2.4.4 Process HTML Response inside Browser....8

2.4.5 Example – HTTP GET Request....8

2.4.6 Example – HTTP POST Request....8

2.5 Ajax with JSON....9

2.6 Compatibility....9

2.7 Restrictions....10

2.8 Debug Support....10

3 Embedded Side Ajax....11

3.1 Embedded Ajax API....11

3.1.1 stxAJAX_Init (fnNotifyHandler)....11

3.1.2 stxAJAX_Tick ( )....11

3.1.3 stxAJAX_CallbackSet (szFileName, fnGenerator, fnParser)....12

3.1.4 stxAJAX_CallbackRemove (szFileName)....12

3.1.5 <USER-GENERATOR-FUNCTION> (pucBuffer, usBufSize, bFirst)....12

3.1.6 <USER-PARSER-FUNCTION> (pucBuffer, usBufSize, bFirst)....13

3.1.7 Adjustable Parameters....14

3.2 Embedded JSON....14

3.2.1 Error Handling, Re-Entrance and Instances....14

3.2.2 ANSI, UTF-8 and Unicode Characters....15

3.2.3 Adjustable Parameters....15

3.2.4 Data Type JSON_VAL_TYPE....16

3.2.5 Data Type JSON_ERR_TYPE....16

3.3 JSON-Generator API....17

3.3.1 stxJSON_GenBegin (pszBuffer, pusBufLen)....17

3.3.2 stxJSON_GenKeyEntry (pszBuffer, pusBufLen, szKey, eType, pvValue)....18

3.3.3 stxJSON_GenArrayEntry (pszBuffer, pusBufLen, eType, pvValue)....19

3.3.4 stxJSON_GenCloseObjArr (pszBuffer, pusBufLen)....19

3.3.5 stxJSON_GenEnd (pszBuffer, pusBufLen)....20

3.3.6 stxJSON_GenRst ( )....20

3.3.7 stxJSON_GenGetErrorState ( )....21

3.4 JSON-Parser API....21
3.4.1 stxJSON_PrsBegin(fnKeyEntry, fnArrayEntry, fnError)....21
3.4.2 stxJSON_PrsParse (szDataPacket, usPacketLength)....22
3.4.3 stxJSON_PrsEnd ( )....22
3.4.4 stxJSON_PrsRst ( )....22
3.4.5 <USER-KEY-ENTRY-FUNCTION> (szKey, eType, pvValue, ucDepth)....23
3.4.6 <USER-ARRAY-FUNCTION> (eType, pvValue, ucDepth)....24
3.4.7 <USER-ERROR-FUNCTION> (ulPosition, cChar)....24

4 Browser Side Ajax....25
4.1 JavaScript Differences in Web Browsers....25
4.2 Validating, Testing and Debugging JavaScript....25
4.3 Compacting JavaScript....25
4.4 SEVENSTAX Ajax Library....25
4.4.1 stxFct_LIB_xxx – Library Initialisation....26
4.4.2 stxFct_ELM_xxx – Browser independent usage of Handles, Objects and DynHTML....26
4.4.3 stxFct_HTML_xxx – Dynamic HTML Code Generation....27
4.4.4 stxFct_XHR_xxx – XML Header Request (XHR) Support....27
4.4.5 stxFct_DBG_xxx – Runtime DebugOut support....28
4.4.6 stxFct_IFR_xxx – I-Frame support....28
4.4.7 stxFct_INPUT_xxx – HMTL Input Field support....28
4.4.8 stxFct_MATH_xxx – Mathematical support....28
4.4.9 stxFct_STYLE_xxx – CSS Style Manipulation....28
4.4.10 stxFct_TIME_xxx – Time Measurement support....28

5 Tools....29
5.1 Validators....29
5.1.1 Jsl.exe....29
5.1.2 Jslint.com....29
5.1.3 Jsonlint....29
5.2 Squeezers / Obfuscators....29
5.2.1 Jsmin.exe....29
5.2.2 Jasob....30
5.3 Browser Plugins....30
5.3.1 Mozilla - FireBug....30
5.3.2 Mozilla - WebDeveloper....30
5.4 Test Webservers....30
5.4.1 Abyss Webserver....30
5.4.2 Apache Webserver....30

6 Change History....31

1 Abstract

AJAX - **A**synchronous **J**avaScript **a**nd **X**ML – is a method to improve dynamic HTML page behaviour *after* the page has been loaded by the browser. It bases on an additional JavaScript-based HTML-Request transferred via TCP-Port 80. Please refer to common literature to retrieve more information about Ajax technologies (e.g. http://www.w3.org/TR/XMLHttpRequest/).

JSON - **J**ava**S**cript **O**bject **N**otation – is a media type to provide structured information for the Internet Community. It is defined in **RFC4627** and defines a very small set of formatting rules in comparison to XML. It uses structures close similarly to C-language structures. It is therefore particularly suitable for embedded applications (see http://www.ietf.org/rfc/rfc4627.txt for details)

SEVENSTAX Ajax is a product part of the SEVENSTAX Internet Suite. It supports dynamic HTML on both sides – Browser-Side and Embedded-Side. It requires and works close together with the SEVENSTAX Embedded Webserver.

This User Manual might be used to get closer to the Ajax technology prepared by SEVENSTAX ready to use for customers. This manual

- lists the provided features,
- shows the needed modules on both sides (embedded/Browser),
- gives a glance behind the structure,
- provides many examples, and
- lists applicable tools

Please note that the SEVENSTAX Reference Application with the Embedded Webserver makes a comprehensive use of Ajax. It is designated to help understanding SEVENSTAX Products, so please feel free to investigate the embedded/browser side code of the application.

Please feel free to contact SEVENSTAX for any question, while trouble shooting, or for any suggestion for improvement.

2 Product Definition

This chapter gives an overview of the features and what is required to use SEVENSTAX Ajax. It is also shown which modules are provided. You will also find an explanation on how it works and a list of compatible browsers.

2.1 Features

SEVENSTAX Ajax incorporates the following modules:

Ajax-C-Interface Module

- Compact C-Module for embedded side
- Ready to use, already connected to SEVENSTAX Webserver
- Easy support of User functions (Callback Handler)
- Supports both HTTP-GET and -POST requests, and responses
- Supports sending and receiving of any User defined file
- Adaptable number of parallel processed Ajax instances and memory consumption
- Ready to use adaption to JSON File
- Supports fragmented files

JSON-C-Library Module

- Complete C-JSON-Library for embedded side
- Based on RFC 4627
- Supports parsing and generating JSON files
- Flexible JSON parser, supporting any JSON data type, objects and array
- Easy to use parser API (User-Callback for any Key, Object or Error)
- Optimized generation of JSON fragments (reduces memory consumption)
- Supports fragmented generation and parsing of files
- Supports UTF-8 conversion

Ajax-JavaScript-Module

- JavaScript-Ajax-Library for browser side
- Easy to integrate into User Web page
- Ready to use API to generate JavaScript HTTP requests (GET and POST methods)
- Supports XML-HeaderRequest (XHR) with Timeout surveillance
- API to receive, validate and parse HTTP responses
- Integrated JSON support (Browser-native or external)
- Easy creation of Dynamic HTML code into user web page
- Adaption layer to support different browsers (no code changes necessary)

- JSLINT conform

2.2 Requirements

The **Developer** should have deeper knowledge about

- generating HTML and CSS code
- designing JavaScript code
- JSON data structure
- SEVENSTAX Webserver product
- some essential tools to validate JavaScript/JSON

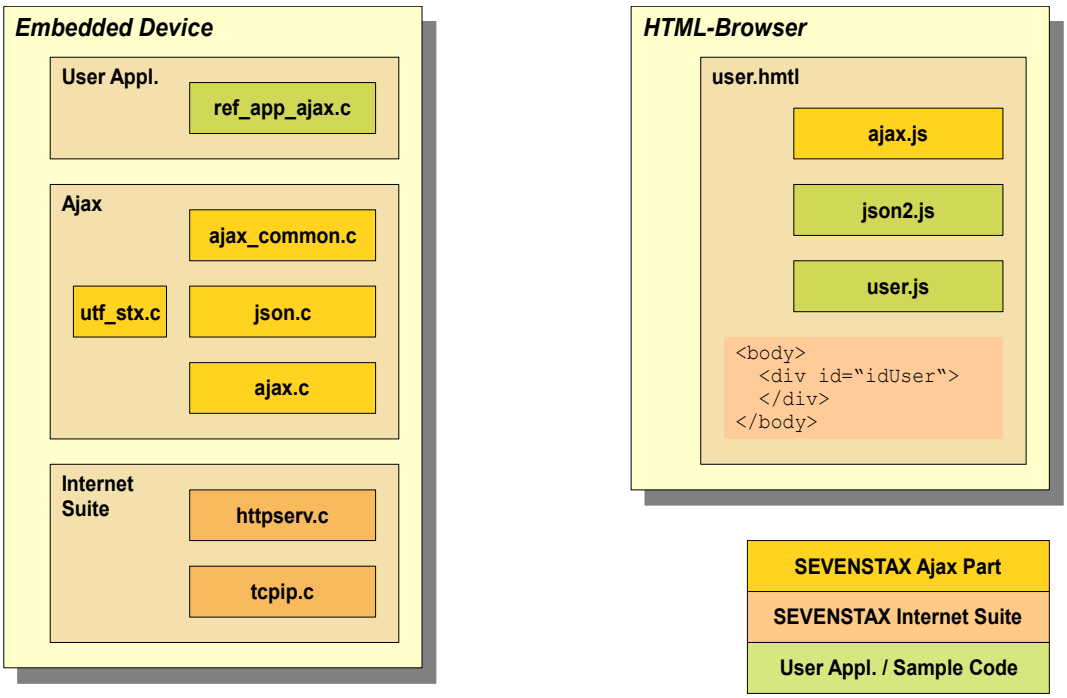
The **Target System** should be provided with

- SEVENSTAX TCP/IP with Webserver
- SEVENSTAX Ajax Product
- enough resources to integrate Web pages and JavaScript code

Please note that – of course – JavaScript must be allowed to use on the **Customer System**. This might not always be the case.

2.3 Ajax Modules

The following graphic gives an overview to the involved SEVENSTAX modules.



On Embedded Side (going from bottom to top), the TCP/IP and Webserver modules provide basic HTTP methods. HTTP requests or files will be transferred to the Ajax-module, to look for User Callback functions, provided by the User Application (e.g. Reference Application). The User Callback

functions might call the JSON-Module to parse/generate JSON files. Some Diagnostic-Helper JSON files (like 'netstat') are provided ready to use in the Ajax-Common module.

On Browser Side, the Web page contains/inserts user defined JavaScript code, which calls functions of the Ajax module to request for e.g. JSON files. Answers of Embedded Side will be validated inside the Ajax module and then inserted into Web page by the user JavaScript code.

2.4 How it works

This chapter describes basic methods on how to use the Ajax functionality for a very common use case: "*The web page contains elements, which are to be updated at runtime. Update is done via a JSON file".*

The following steps have to processed for a complete update cycle:

1. Send a GET request from the browser to the device
2. Process the GET request on the device
3. Generate and send HTTP response from the device to the browser
4. Process the HTTP response inside the browser

2.4.1 Send HTTP Request from Browser ⇨ Device

1. Prepare HTML elements with a dedicated container, which will receive the dynamically generated HTML content.
2. Prepare User JavaScript code to stimulate a GET request on a special event (e.g. Click or Timer event), to update the container.
3. Call the Ajax-Lib-Function *stxFct_XHR_GetFile()* with requested *filename* and dedicated *JavaScript-User Callback* function.
4. The JavaScript engine of the browser will open a TCP connection, transfer the request and notify the User Callback for any event concerning this request.

2.4.2 Process HTTP Request on Device

1. SEVENSTAX Embedded TCP/IP Stack and Webserver are initialized to listen on HTTP server port 80 via *stxTCP_Listen()*.
2. User Callback functions are registered to the Ajax module via *stxAJAX_CallbackSet()* to handle the requested filename
3. The TCP/IP Stack will deliver the filename of the GET request to the Webserver.
4. The Webserver will not find the filename inside its resources and will ask the Ajax.
5. Ajax will find the registered filename and call the associated User Callback.

2.4.3 Send HTTP Response from Device ⇨ Browser

1. The User Callback will allocate some memory and generate a JSON file with the help of the stxJSON_xxx() functions.
2. After returning from the User Callback, the Webserver and TCP/IP-Stack will send this generated buffer to the browser.
3. The User Callback is called for multiple JSON file fragments as long as it does return AJAX_GEN_PAKET.

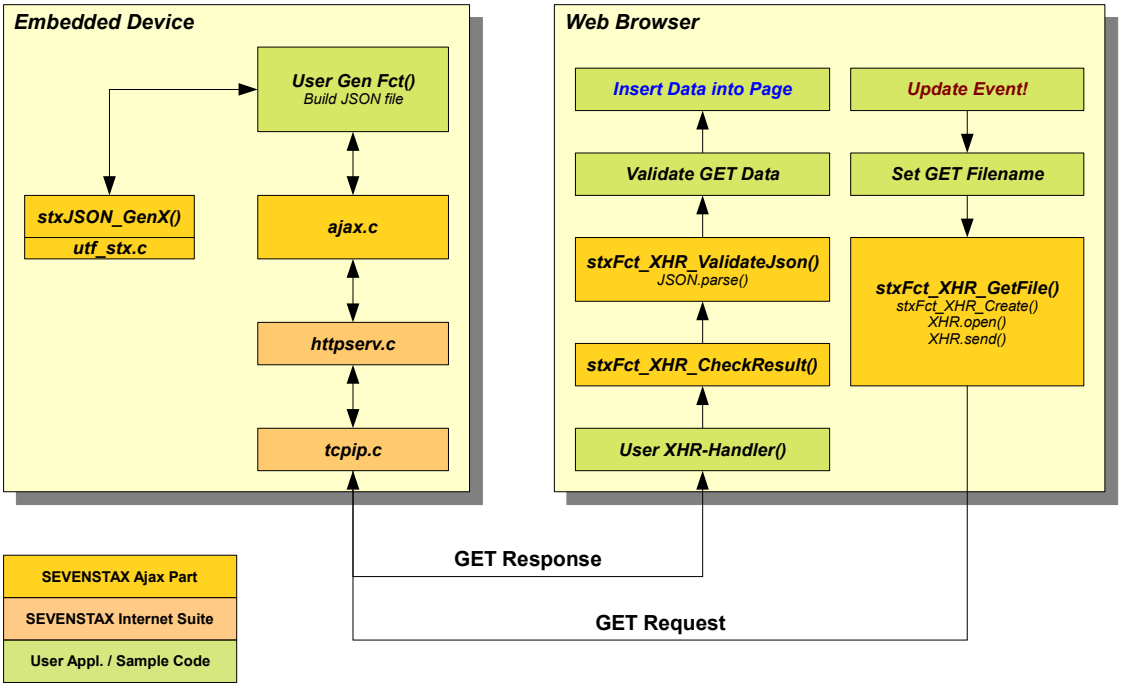
4. If the file is completed, the User Callback returns AJAX_GEN_READY. This closes the TCPconnection.

2.4.4 Process HTML Response inside Browser

1. The JavaScript Engine of the browser will receive and collect the delivered JSON file fragments. If the file is completed, the engine will call the registered JavaScript-Callback function of this GET request (see above)
2. The User JavaScript-Callback first calls the *stxFct_XHR_CheckResult()*, to retrieve the current state of the GET request.
3. If the result is ok, it then calls the *stxFct_XHR_ValidateJson()* to validate the expected JSON file. The result is a JSON object.
4. If the JSON object is valid, it then checks the content of the JSON file for any semantic errors. This is very use case specific.
5. If the JSON content is ok, the User Script extracts the Keys, Arrays and Objects and converts it into formatted HTML code with the help Ajax-Lib-Fct. like *stxFct_HTML_TableAdd()*.
6. The generated HTML code can now be inserted with e.g. *stxFct_ELM_HtmlSetContent()* into the container mentioned above.
7. Now, the update cycle is completed.

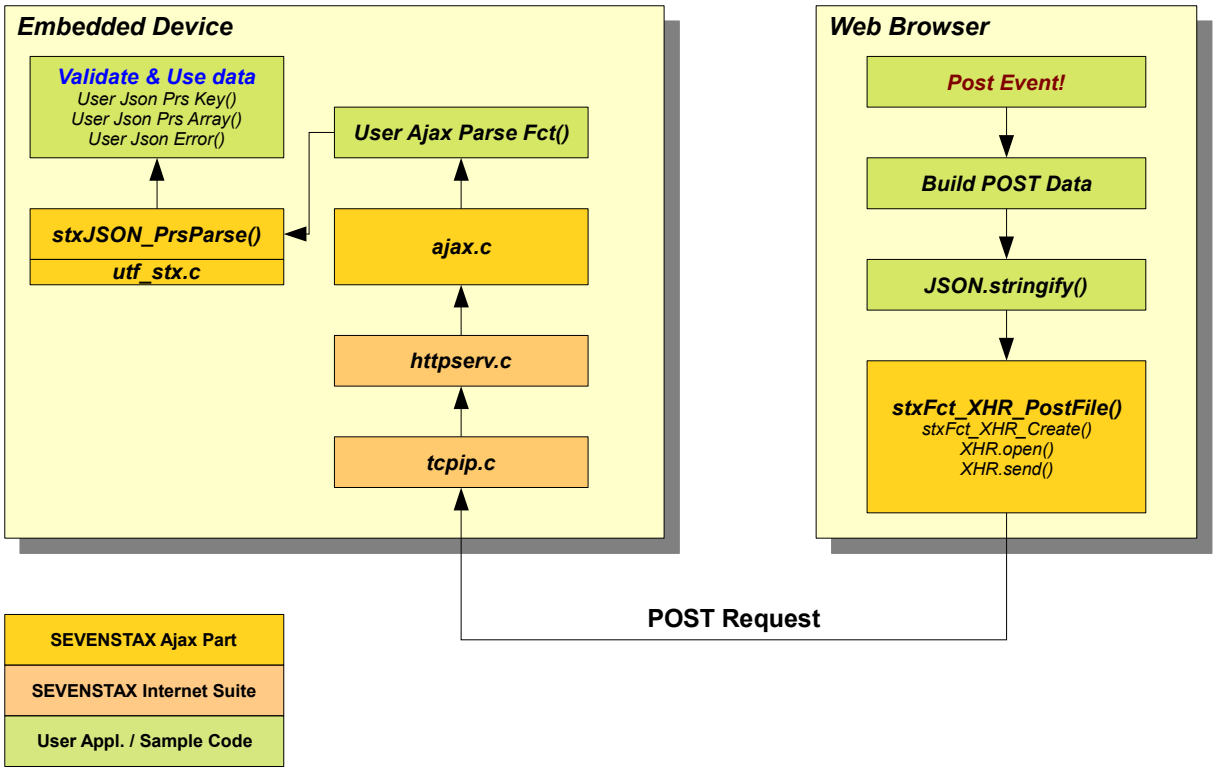
2.4.5 Example – HTTP GET Request

The graphic below shows the typical program flow for a HTTP GET request with a GET response of the device. The flow starts with an Update Event (e.g. timer to refresh the page) and ends with fresh data inserted into web page.



2.4.6 Example – HTTP POST Request

The graphic below shows the typical program flow for a HTTP POST request without a response. The flow begins with Post event (e.g. a Submit-Button) and ends with validate fresh user data in the embedded device.



Please note, that a HTTP POST request can optionally be answered with a HTTP GET Response. This can be useful to update the devices accepted data immediately in the web page.

2.5 Ajax with JSON

SEVENSTAX Ajax supports any kind of user file requested by the browser. But JSON files are a very convenient and reliable way, to support structured data.

On Embedded Side, SEVENSTAX Ajax provides the customer with a complete JSON C-Library, especially developed for small embedded systems (see chapters 3.3 and 3.4 for details).

On Browser Side, newer releases of Web browsers (e.g. Mozilla FireFox 3.5, Microsoft Internet Explorer 8.0) support JSON as “native”. This means, that no JavaScript has to be inserted into the Webpage, the JSON functions are already inside the browser.

Older browser releases do not support JSON as native. To enable JSON for these browsers, a dedicated Lib file can be inserted into the web page. A ready to use and free of charge JSON lib can be found on the Web page of the JSON inventor: http://www.json.org/json2.js

2.6 Compatibility

SEVENSTAX Ajax is approved to be usable with the following Internet browsers:

- Mozilla FireFox V3.0, V3.5.x,
- Internet Explorer V6, V7, V8
- Opera V10.10
- Safari V3.1.2

2.7 Restrictions

As SEVENSTAX Ajax is a very compact implementation for small embedded devices, the following restrictions apply to this product:

1. The product modules currently support the structured JSON file format only as ready to use, inclusive parser and generator.
2. XML parser and generator are available as separate product modules, but they are currently not integrated inside the Ajax product.
3. Any kind of structured or non structured data can be transferred via the SEVENSTAX Ajax, as far as the data is parsed and generated by user application code on embedded and browser side.
4. The JSON parser supports one active instance only. As long as the received JSON file is unfragmented, this is no restriction, because the HTTP server will not be interrupted by other received JSON files.

2.8 Debug Support

As all SEVENSTAX products, a printf() interface is supported to deliver any kind of info, warning error. This might be usable at development process only and enhances test phase speed.

Please enable (#define as 1) one/all of the following switches in 'features.h':

- DEBUG_STX – main switch to enable DebugOuts
- AJAX_DEBUG – enables details of Ajax module activity and warning
- JSONGEN_DEBUG – enables details of JSON generator only
- JSONPRS_DEBUG – enables details of JSON parser only

If not enabled, no code is generated and ROM resources are saved.

Note: Especially the JSONxxx_DEBUG debug switches might generate a large amount of debug lines which could slow down the application.

3 Embedded Side Ajax

Embedded Side Ajax means those SEVENSTAX product parts (libraries) supporting Ajax functionality in the embedded device, designed in programming language ANSI-C.

For the Browser Side Ajax, please refer to chapter 4.

3.1 Embedded Ajax API

The Ajax module has two interfaces:

- to the upper layer, which is the user application, and
- to the lower layer, which is the SEVENSTAX Webserver

The lower layer interface is already connected to the Webserver, nothing has to be done other than enabling **#define AJAX_SUPPORTED == 1** to enable the Ajax module.

So the following chapters describe the upper layer to the user application only.

3.1.1 stxAJAX_Init (fnNotifyHandler)

Description

Initializes the AJAX module and all instances. Clears all current Ajax instances and therefore all Ajax transmit buffers.

Parameter

PROTOCOL_NOTIFY_HANDLER handler
Not yet used.

Return Value

-

Comment

Must be called first, before using Ajax. File specific user callback functions must be registered with *stxAJAX_CallbackSet ()* after calling this function, to let Ajax work.

3.1.2 stxAJAX_Tick ( )

Description

This function keeps AJAX alive.

Parameter

-

Return Value

void

Comment

Not yet used.

3.1.3 stxAJAX_CallbackSet (szFileName, fnGenerator, fnParser)

Description

Allow the user to add file specific handlers for generator and parser for the given filename. Ajax will call these callback functions whenever POST / GET requests occur for this file.

Parameter

STRING_stx szFileName (mandatory)
Filename for which the following callback functions should be used.

void FPTR_stx fnGenerator
Reference to user callback function, which is able to generate the requested file.

void FPTR_stx fnParser
Reference to user callback function, which is able to parse the requested file.

Return Value

UINT8_stx
Number of initialized Ajax instance, which is now connected to that file.

UINT8_stx_MAX
If any error occurs, e.g. no free Ajax instance or wrong parameters.

Comment

One – but not both - of the given fnGenerator / fnParser callback is optional.

Note: If the requested file is already found inside static Web resources of the Webserver (ROM table or external (flash) file system), the handler function will NOT be called!

3.1.4 stxAJAX_CallbackRemove (szFileName)

Description

Allow user to remove the file specific handlers (generator/parser). Ajax will then no longer support that file.

Parameter

STRING_stx szFileName
Name of file no longer to be supported.

Return Value

UINT8_stx
Number of cleared Ajax instance, if found

UINT8_stx_MAX
If not found any any error occurs.

Comment

-

3.1.5 <USER-GENERATOR-FUNCTION> (pucBuffer, usBufSize, bFirst)

Description

Generator function prototype to handle callbacks of Ajax. Will be called for any **HTTP-GET** request to the registered file (see *stxAJAX_CallbackSet ()*).

This function has to be designed by user application to support generation of Ajax files of any kind (not only JSON). It will be called multiply with different parameters by Ajax to start, continue, and finish file GET transfer via Ajax.

Parameter

UINT8_stx FPTR_stx pucBuf
Reference to Ajax send buffer. Callback should insert generated data here. If *pucBuf* == *NULL_stx*, Ajax signals, that nothing more is to be done, because of completion or closing by error. This could be used e.g. to reset a generator.

UINT16_stx usBufSize
Contains the size of delivered Ajax send buffer *pucBuf* in bytes. User callback must take care not to exceed this buffer.

BOOL_stx bFirst
Ajax sets this flag to TRUE_stx, if this is the very first time the callback is called. This can be used to start the specific generator or to stimulate a background process to update data to be sent.

Return Value

Predefined enum type AJAX_GEN_STATE:

```
typedef enum
{
    AJAX_GEN_WAIT,      // Generator not yet ready          -> Webserver tries again later
    AJAX_GEN_PAKET,     // Generator has a new packet       -> Webserver transfers data
    AJAX_GEN_READY,     // Generator has finished Ajax data -> Webserver closes connection
    AJAX_GEN_FAIL,      // Generator failed to start        -> Webserver closes connection
    AJAX_GEN_RST        // Generator reset
} AJAX_GEN_STATE;
```

Comment

AJAX_GEN_WAIT can be used by application to delay the Ajax file transfer for any part. This might be useful if a background process is not yet ready to deliver the requested data of while the generator function is currently busy (e.g. JSON generator).

Ajax will continuously call this callback until the callback returns **AJAX_GEN_READY**. Each time it returns **AJAX_GEN_PAKET**, a TCP/HTTP packet will be send to remote as part of the requested file.

3.1.6 <USER-PARSER-FUNCTION> (pucBuffer, usBufSize, bFirst)

Description

Parser function prototype to handle callbacks of Ajax. Will be called for any **HTTP-POST** request to the registered file (see *stxAJAX_CallbackSet ()*).

This function has to be designed by user application to support parsing of Ajax files of any kind (not only JSON). It will be called multiply with different parameters by Ajax to start, continue, and finish POST file transfer via Ajax.

Parameter

UINT8_stx FPTR_stx pucBuf
Reference to Ajax receive buffer. Callback should parse through the complete buffer for *usBufSize bytes.* If *pucBuf* == *NULL_stx*, Ajax signals, that nothing more is to be done, because of completion or closing by error. This could be used e.g. to reset a parser.

UINT16_stx usBufSize
Contains the size of delivered Ajax receive buffer *pucBuf* in bytes. User callback must take care not to read behind this buffer.

BOOL_stx bFirst
Ajax sets this flag to TRUE_stx, if this is the very first time the callback is called. This can be used to start the specific parser or to stimulate a background process to pass the received data.

Return Value

void

Comment

Because the HTTP servers currently work in streaming manner, each call of this callback function will contain only one extracted byte of the HTTP-POST request. This might be changed in future versions.

3.1.7 Adjustable Parameters

The following parameters are pre-defined in 'ajaxdefs.h' with default values. Please do not change this file. As for all SEVENSTAX Internet Suite Protocols, the parameters can be set up for user requirements in '**features.h**', which will override the default settings.

AJAX_BUFFSIZE **default: 1000 valid: 100..n**

Defines the size in bytes for one Ajax instance send buffer to construct Ajax files. The complete memory consumption is calculated by AJAX_BUFFSIZE x AJAX_INSTANCES.

If the file generated by the user callback generator is larger than AJAX_BUFFSIZE, it must be splitted by the user callback into several fragments and transfered in parts. This is supported by the AJAX_GEN_PAKET / AJAX_GEN_READY return values (see *stxAJAX_CallbackSet ()*).

AJAX_INSTANCES **default: 2 valid: 1..n**

Defines the number of concurrent possible Ajax transfer and send buffers. The complete memory consumption is calculated by AJAX_BUFFSIZE x AJAX_INSTANCES.

Ajax transfers might be split into several parts and there might block one Ajax instance for a longer time. If another Ajax HTTP request appears, Ajax tries to find a free instance.

AJAX_FUNCTABLESIZE **default: 10 valid: 1..n**

Defines the maximum number of supported Ajax user files – therefore defines size of the Ajax function table.

Each function table entry contains a filename and a user callback function for the generator and parser. (see *stxAJAX_CallbackSet ()* for details).

3.2 Embedded JSON

3.2.1 Error Handling, Re-Entrance and Instances

The JSON generator functions take care not to exceed the given buffer. They will return error codes, if any problem occurs. The user should check the return value of each call of a *stxJSON_GenXXX()*, to detect problems. Additionally *stxJSON_GenGetErrorState()* is prepared to retrieve error information.

The JSON generator and parser currently are non re-entrant and support a single generator / parser instance only! This means, only one generation and one parser process is allowed at a time. The user first has to check, whether the generator is free to use. This can implicitly be done when calling *stxJSON_GenBegin()* and checking the return value.

3.2.2 ANSI, UTF-8 and Unicode Characters

For string values, JSON defines a valid character set (please see http://www.json.org/). As long as the JSON file contains characters between 0x20..0x7f for any kind of strings or key values, there is no need to convert anything.

But anything outside this area is to be converted to keep a JSON file valid. Any UTF-8 character is supported implicitly by the JSON format and is not converted. Control characters < 0x20 are directly supported by JSON with the help of ESC-sequence, like \b, \f, \n, \r, \t. JSON reserved characters have to be escaped too, like \", \\ , \/ and \u. Any Unicode character has to be represented with the ESC sequence “\uxxxx.

The SEVENSTAX **JSON generator** supports these requirements with the following rules for strings delivered by the user application:

1. The ANSI control characters like **\b, \f, \n, \r, \t** are implicitly converted into strings containing the escaped character pattern, e.g. ”**\b”, “\f”, “\n”, “\r”, “\t”.** Any other character **<0x20** is converted into a Unicode sequence like **“\u001f”** instead.
2. The reserved characters **", \ , /,** are implicitly escaped to strings containing **“\””, “\\”, “\/”.**
3. Characters **> 0x7f** are expected to be **UTF-8-Code** and are not changed in any way. The user application is responsible to deliver strings in UTF-8-Code. This can be done with the help of the '*utf_stx.c*' library, which is part of SEVENSTAX Ajax.
4. The user application **must not** deliver strings containing **Unicode** sequences like “\u2128”. The Generator would convert this into “\\u2128”, which is no more Unicode. Please exclusively use UTF-8 inside the user application, which should be sufficient for any language and works with a minimum of memory overhead.

The SEVENSTAX **JSON parser** supports the requirements with the following rules for received strings, which are delivered to the user application:

1. Escaped character pattern, e.g. ”**\b”, “\f”, “\n”, “\r”, “\t”** are implicitly converted into the ANSI control characters **\b, \f, \n, \r, \t**. Any Unicode sequence **< \u0020** is too converted into the adequate ANSI control characters.
2. Escaped strings containing **“\””, “\\”, “\/”** are implicitly converted into the characters **", \ , /**.
3. Any characters **> 0x7f** are expected to be **UTF-8-Code** and are not changed in any way. The user application is responsible to handle these strings. This can be done with the help of the '*utf_stx.c*' library, which is part of SEVENSTAX Ajax.
4. Any **Unicode** sequence **> \u001f** is not converted. The user application is responsible to handle this sequence. Please prefer to use UTF-8 inside the Web application, which should be sufficient for any language and works with a minimum of memory overhead.

Please note, that SEVENSTAX Ajax provides a **UTF-8-Converter** module 'utf_stx.c' to convert numbers between a Unicode and UTF-8.

3.2.3 Adjustable Parameters

The following parameters are pre-defined in 'jsondefs.h' with default values. Please do not change this file. As for all SEVENSTAX Internet Suite Protocols, the parameters can be set up for user requirements in '**features.h**', which will override the default settings.

JSON_GEN_VAL_LEN **default: 50** **valid: 1..n**

Defines the size of the temporary string buffer in bytes to generate a value. As all values are converted into strings, please take care for this setting. The JSON API assures not to exceed the buffer.

JSON_PRS_KEY_LEN **default: 50** **valid: 1..n**

Defines the size of the temporary buffer in bytes to extract and deliver a key name string. The key name string buffer will be delivered to the user callback function.

JSON_PRS_VAL_LEN **default: 50** **valid: 1..n**

Defines the size of the temporary buffer in bytes to extract and deliver a value of any kind. The value buffer will be delivered to the user callback function.

JSON_RECURSION_MAX **default: 10** **valid: 1..n**

Defines the maximum object complexity of a JSON file for parsing & generating. The object complexity is defined by the number nested braces for objects “{..}” and arrays “[..]”. parser and generator will stop working and detect an error, if the currently processed JSON file exceeds this value.

3.2.4 Data Type JSON_VAL_TYPE

This enum data type defines all supported types for JSON <VALUES>:

```
/* JSON data type defines for public API (see http://json.org/ for details)*/
typedef enum
{
    JSON_VAL_TYPE_STRING,       // string
    JSON_VAL_TYPE_UINT_8,       // unsigned char
    JSON_VAL_TYPE_UINT_16,      // unsigned short
    JSON_VAL_TYPE_UINT_32,      // unsigned long
    JSON_VAL_TYPE_SINT_8,       // signed char
    JSON_VAL_TYPE_SINT_16,      // signed short
    JSON_VAL_TYPE_SINT_32,      // signed long
    JSON_VAL_TYPE_FLOAT,        // float
    JSON_VAL_TYPE_BOOL,         // boolean
    JSON_VAL_TYPE_NULL,         // 'NULL' type value
    JSON_VAL_TYPE_OBJECT,       // object '{..}'
    JSON_VAL_TYPE_ARRAY,        // array '[ ..]'
    JSON_VAL_TYPE_END           // indicates 'end of' value/object/array
} JSON_VAL_TYPE;
```

JSON_VAL_TYPE_STRING .. **JSON_VAL_TYPE_NULL** might appear as simple types with a pair of “<KEYNAME>:<VALUE>;”, or as part of an object or array.

JSON_VAL_TYPE_OBJECT just opens a subsequent **“{ .. }”** block for more (structured) data. Be sure, to always close such a structure with *stxJSON_GenCloseObjArr()*.

JSON_VAL_TYPE_ARRAY opens a subsequent **“[..]”** array data of equal data type. Be sure, to always close such a structure with *stxJSON_GenCloseObjArr()*.

JSON_VAL_TYPE_END is used for parsing only, as a signal to user callback, that an array/object is just being closed.

3.2.5 Data Type JSON_ERR_TYPE

This enum data type defines error state of JSON generator:

```
/* JSON error type defines, generated from stxJSON_GenBegin() until stxJSON_GenEnd() */
typedef enum
{
    JSON_ERR_OK,            // no error occured
    JSON_ERR_INVPAR,        // invalid parameters
    JSON_ERR_BUSY,          // parser/generator instance still busy
    JSON_ERR_STR2LONG,      // key/value too long
    JSON_ERR_INVTYPE,       // invalid type
    JSON_ERR_REC_EXC,       // recursion counter exceeded
    JSON_ERR_BUF_EXC        // remaining buffer exceeded
} JSON_ERR_TYPE;
```

3.3 JSON-Generator API

The JSON generator works with dedicated API functions, each specialized to add one JSON element per call - like a key, an object or array.

Please note: The JSON generator currently is non re-entrant and supports a single instance only! This means, only one generation process is allowed at a time. Any concurrent try to start the generator will be blocked.

This is an overview of the supported generator functionality.

- To open the JSON file, use *stxJSON_GenBegin().*
- To close the JSON file, use *stxJSON_GenEnd()*.
- To add a pair of key and value, use *stxJSON_GenKeyEntry()*
- To open an array of values, also use *stxJSON_GenKeyEntry()*
- To add an array element, use *stxJSON_GenArrayEntry()*.
- To close an array, use *stxJSON_GenCloseObjArr()*.
- To open an object, use *stxJSON_GenKeyEntry() with JSON_VAL_TYPE_OBJECT.*
- To add an object element, use the standard *stxJSON_GenKeyEntry()*
- To close an object, use *stxJSON_GenCloseObjArr().*

This is an example of a JSON file, generated by SEVENSTAX JSON generator (of course without formating Tabs or Spaces):

```
{
    "szFileName": "ethlogcntrl.json",
    "Flags": {
        "fEnableARP": true,
        "fEnableUDP": false,
        "fEnableTCP": true,
        "fEnableIPFilter": false,
        "fEnableMACFilter": false
    },
    "IPAddrMask": [ 255, 255, 255, 255 ],
    "MACAddrMask": [ 255, 255, 255, 255, 255, 255 ]
}
```

3.3.1 stxJSON_GenBegin (pszBuffer, pusBufLen)

Description

Starts generating a new JSON file, inserts the open char.

This is the starting point of any JSON generator process. It checks current state, initializes the process and inserts the first valid JSON character "{".

Parameter

STRING_stx FPTR_stx pszBuffer
Reference to address of generator buffer to insert JSON data. Will be incremented with count of bytes, which are inserted into the buffer by this function. Will be returned for further use.

UINT16_stx FPTR_stx pusBufLen
Reference to variable which contains the remaining size of *pszBuffer* in bytes. Before adding any kind of data, this value will be checked. Insertion will be prevented, if the new data does not fit into remaining buffer. Will be decremented with count of bytes, which are inserted into the buffer by this function. Will be returned for further use.

Return Value

BOOL_stx
TRUE_stx if everything worked well. FALSE_stx if the remaining buffer or current state does not allow insertion of given data.

Comment

Any call of this function must be completed with a call to *stxJSON_End()*.

3.3.2 stxJSON_GenKeyEntry (pszBuffer, pusBufLen, szKey, eType, pvValue)

Description

Adds a new complete <KEY> to the JSON file, or just opens an array for <KEY>. Inserts a comma, too, if previous object elements exist.

If the eType is a simple data type (see JSON_VAL_TYPE), the given pvValue is directly inserted and the key is closed.

If the eType is an array, just the <KEY> and the array-open "[" is inserted. Subsequent calls of *stxJSON_GenArrayEntry()* are needed.

Parameter

STRING_stx FPTR_stx pszBuffer
Reference to address of generator buffer to insert JSON data. Will be incremented with count of bytes, which are inserted into the buffer by this function. Will be returned for further use.

UINT16_stx FPTR_stx pusBufLen
Reference to variable which contains the remaining size of *pszBuffer* in bytes. Before adding any kind of data, this value will be checked. Insertion will be prevented, if the new data does not fit into remaining buffer. Will be decremented with count of bytes, which are inserted into the buffer by this function. Will be returned for further use.

STRING_stx szKey
The string containing the name of the key to be added.

JSON_VAL_TYPE eType
One of predefined data types JSON_VAL_TYPE. If *eType* is JSON_VAL_TYPE_ARRAY, just the <KEY> and the array-open "[" is added.

void FPTR_stx pvValue
Reference to a buffer, which contains the data defined by *eType*. Ignored, if *eType* is JSON_VAL_TYPE_ARRAY.

Return Value

BOOL_stx
TRUE_stx if everything worked well. FALSE_stx if the remaining buffer or current state does not allow insertion of given data.

Comment

In case of JSON_VAL_TYPE_STRING, please take care about the valid characters defined in JSON (see chapter 3.2.2, ANSI, UTF-8 and Unicode Characters for details).

3.3.3 stxJSON_GenArrayEntry (pszBuffer, pusBufLen, eType, pvValue)

Description

Adds an array entry. Inserts the value only (no key required), but the *eType* must be given too to handle the given void ptr correctly. Inserts a Comma too, if previous array elements exist.

The eType can be any data type, key, array or object (see JSON_VAL_TYPE).

Parameter

STRING_stx FPTR_stx pszBuffer
Reference to address of generator buffer to insert JSON data. Will be incremented with count of bytes, which are inserted into the buffer by this function. Will be returned for further use.

UINT16_stx FPTR_stx pusBufLen
Reference to variable which contains the remaining size of *pszBuffer* in bytes. Before adding any kind of data, this value will be checked. Insertion will be prevented, if the new data does not fit into remaining buffer. Will be decremented with count of bytes, which are inserted into the buffer by this function. Will be returned for further use.

JSON_VAL_TYPE eType
One of predefined data types JSON_VAL_TYPE.

void FPTR_stx pvValue
Reference to a buffer, which contains the data defined by *eType*. Ignored, if *eType* is JSON_VAL_TYPE_ARRAY.

Return Value

BOOL_stx
TRUE_stx if everything worked well. FALSE_stx if the remaining buffer or current state does not allow insertion of given data.

Comment

The array must have previously been opened with *stxJSON_GenKeyEntry()*. The array must be closed after adding all entries with *stxJSON_GenCloseObjArr()*.

In case of JSON_VAL_TYPE_STRING, please take care about the valid characters defined in JSON (see chapter 3.2.2, ANSI, UTF-8 and Unicode Characters for details).

3.3.4 stxJSON_GenCloseObjArr (pszBuffer, pusBufLen)

Description

Closes an array or object.

Automatically detects last opened structure (array/object) and adds adequate closing char **“]”** or **“}”**, updates internal object depth, too.

Parameter

STRING_stx FPTR_stx pszBuffer
Reference to address of generator buffer to insert JSON data. Will be incremented with count of bytes, which are inserted into the buffer by this function. Will be returned for further use.

UINT16_stx FPTR_stx pusBufLen
Reference to variable which contains the remaining size of *pszBuffer* in bytes. Before adding any kind of data, this value will be checked. Insertion will be prevented, if the new data does not fit into remaining buffer. Will be decremented with count of bytes, which are inserted into the buffer by this function. Will be returned for further use.

Return Value

BOOL_stx
TRUE_stx if everything worked well. FALSE_stx if the remaining buffer or current state does not allow insertion of given data.

Comment

Must be called after completed array or object data.

3.3.5 stxJSON_GenEnd (pszBuffer, pusBufLen)

Description

Closes the JSON file by adding the closing **"}"** char to buffer.

As an error detection, additionally checks for still open objects or array and closes them too, to keep the JSON file valid.

Parameter

STRING_stx FPTR_stx pszBuffer
Reference to address of generator buffer to insert JSON data. Will be incremented with count of bytes, which are inserted into the buffer by this function. Will be returned for further use.

UINT16_stx FPTR_stx pusBufLen
Reference to variable which contains the remaining size of *pszBuffer* in bytes. Before adding any kind of data, this value will be checked. Insertion will be prevented, if the new data does not fit into remaining buffer. Will be decremented with count of bytes, which are inserted into the buffer by this function. Will be returned for further use.

Return Value

BOOL_stx
TRUE_stx everything worked well. FALSE_stx if the remaining buffer or current state does not allow insertion of given data.

Comment

-

3.3.6 stxJSON_GenRst ( )

Description

Resets the JSON generator.

Parameter

-

Return Value

-

Comment

Can be called before starting the generation process. Should be called in any error case, to ensure a valid restart if the next JSON file has to generated.

3.3.7 stxJSON_GenGetErrorState ( )

Description

Returns last JSON generator error state as element of JSON_ERR_TYPE.

Parameter

-

Return Value

JSON_ERR_TYPE
Last JSON generator error value.

Comment

Might be useful for debugging purpose.

3.4 JSON-Parser API

The JSON parser works as a single function with streaming parser technology. Although the parser is able to parse through larger data blocks, it will be called by the Embedded Webserver with single bytes, detected while parsing through the received HTTP-POST Request.

Please note: The JSON generator currently is non re-entrant and supports a single instance only! This means, only one parsing process is allowed at a time. Any concurrent attempt to start the generator will be blocked.

3.4.1 stxJSON_PrsBegin(fnKeyEntry, fnArrayEntry, fnError)

Description

Prepares to start a parsing process. Registers user callback functions to handle parsing results for the current parsing process

Must be called before calling *stxJSON_PrsParse()*. This can be done, if the Ajax user callback function to handle POST requests (<USER-PARSER-FUNCTION>) is called with the parameter *bFirst == TRUE_stx*. Must be succeeded by a call of *stxJSON_PrsEnd()* at the end of the parsing process.

Parameter

JSON_PRS_CB_KEY_ENTRY fnKeyEntry
User callback function to handle any pair of <KEY> and <VALUE> inside a JSON file.

JSON_PRS_CB_ARRAY_ENTRY fnArrayEntry
User callback function to handle an detected array entry.

JSON_PRS_CB_ERROR fnError
User callback function to handle an detected errors..

Return Value

BOOL_stx
TRUE_stx if parser is free for use and all callback functions valid. *FALSE_stx* in any error case.

Comment

Each user defined JSON file might be supported with a different set of specialized user callback functions to handle JSON keys, array and errors.

3.4.2 stxJSON_PrsParse (szDataPacket, usPacketLength)

Description

Calls the SEVENSTAX JSON streaming parser for defined number characters.

The parser internally will call the user callback functions (registered for the current JSON file at *stxJSON_PrsBegin()*) for any detected key, array or error. The JSON parser saved the internal state of the previous call, so that this function can be called for any fragment of a JSON file.

Parameter

STRING_stx szDataPacket
Reference to buffer with JSON string to be parsed.

UINT16_stx usPacketLength
Number of character inside the delivered *szDataPacket.*

Return Value

BOOL_stx
TRUE_stx if everything went well. *FALSE_stx* in any error case.

Comment

-

3.4.3 stxJSON_PrsEnd ( )

Description

Finishes the parsing process. Frees the parser for further use.

Parameter

-

Return Value

BOOL_stx
TRUE_stx if everything went well. *FALSE_stx* in any error case.

Comment

Should be called if user application detected end of Ajax POST request.

3.4.4 stxJSON_PrsRst ( )

Description

Resets the JSON parser.

Parameter

-

Return Value

-

Comment

Can be called before starting the parser process. Should be called in any error case, to ensure a valid restart if the next JSON file to parsed.

3.4.5 <USER-KEY-ENTRY-FUNCTION> (szKey, eType, pvValue, ucDepth)

Description

User application function prototype to handle a pair of <KEY> and <VALUE> inside the JSON file.

This function has to be designed by user application to support all keys defined for that JSON file. It must previously be registered to the JSON parser (see *stxJSON_PrsBegin()*). It will be called for by JSON parser while processing data inside *stxJSON_PrsParse()*.

Parameter

STRING_stx szKey
String, containing the <KEY> of the detected data.

JSON_VAL_TYPE eType
Detected data type, any of JSON_VAL_TYPE.

void FPTR_stx pvValue
Reference to extracted data. Before using this data, a typecast to JSON_VAL_TYPE - adequate C-data type must be done to prevent application errors.

UINT8_stx ucDepth
Current parser depth, delivered for diagnostic purpose. Every opened object / array increases the *ucDepth*, so the user callback is able to additionally check the syntax.

Return Value

-

Comment

JSON syntax checking is already done by the JSON parser. But semantic check (validator) for the delivered key/value has to be done by this user callback function: It is strongly recommended to use a much as possible delivered information to validate the data:

eType should be used first to check/distinguish the expected data.

szKey must be used to distinguish the expected data. Please note, that this might be invalid, if *eType* is JSON_VAL_TYPE_END.

pvValue should be checked at least to be a valid pointer. Please note, that it is invalid, if the *eType* is JSON_VAL_TYPE_OBJECT, JSON_VAL_TYPE_ARRAY, JSON_VAL_TYPE_END. If *eType* is JSON_VAL_TYPE_STRING, the string length should be checked too, before using.

ucDepth can be used to check that the expected data hierarchy is correct.

3.4.6 <USER-ARRAY-FUNCTION> (eType, pvValue, ucDepth)

Description

User application function prototype to handle an <VALUE> detected inside an array in the JSON file.

This function has to be designed by user application to support all array elements defined for that JSON file. This function must previously be registered to the JSON parser (see *stxJSON_PrsBegin()*). It will be called for by JSON parser while processing data inside *stxJSON_PrsParse()* inside an opened array.

Parameter

JSON_VAL_TYPE eType
Detected data type, any of JSON_VAL_TYPE.

void FPTR_stx pvValue
Reference to extracted data. Before using this data, a typecast to JSON_VAL_TYPE - adequate C-data type must be done to prevent application errors.

UINT8_stx ucDepth
Current parser depth, delivered for diagnostic purpose. Every opened object / array increases the *ucDepth*, so the user callback function is able to additionally check the syntax.

Return Value

-

Comment

Please see <USER-KEY-ENTRY-FUNCTION> for rules and on how to use this function.

3.4.7 <USER-ERROR-FUNCTION> (ulPosition, cChar)

Description

User application function prototype to handle an detected errors while parsing the JSON file, commonly for diagnostic purpose.

This function has to be designed by user application to support any kind of error. This function must previously be registered to the JSON parser (see *stxJSON_PrsBegin()*). It will be called for by JSON parser while processing data inside *stxJSON_PrsParse()*.

Parameter

UINT32_stx ulPosition
The current parser position inside the JSON file, as an offset of bytes since beginning of file, at which the error is detected.

CHAR_stx cChar
The character at the *ulPosition*, at which the error is detected.

Return Value

-

Comment

For diagnostic purpose only.

4 Browser Side Ajax

Browser Side Ajax means those SEVENSTAX product parts (libraries) supporting Ajax functionality in a common Web browser like Firefox, Internet Explorer, Safari or Opera, designed in programming language JavaScript.

For the Embedded Side Ajax, please refer to chapter 3.

4.1 JavaScript Differences in Web Browsers

JavaScript is natively supported by most Web browsers. But there are several implementation differences, from manufacturer to manufacturer and from release version to release version.

To support all these differences in a common way, SEVENSTAX made special effort to abstract the browsers object model.

Please exclusively use the ***stxFct_ELM_xxx()*** functions to retrieve information about elements inside the browser page!

4.2 Validating, Testing and Debugging JavaScript

A JavaScript engine a browser simply stops executing whenever an JavaScript error is detected. Therefore, JavaScript should always be **syntactical checked** (like C-Lint) before use (see chapter 5.1, **Validators** for details).

Before using JavaScript Web Applications on embedded systems, we strongly recommend to **test** the complete Website on a **Standard Webserver** (see 5.4, Test Webservers for details).

To improve debugging features SEVENSTAX include a special **DebugOut** features to Ajax, similar to DebugOut functionality at embedded C-Side: Any HTML- container can be used, to printf() runtime information of the Ajax-Lib or user application to.

Please use *stxFct_DBG_xxx()* functions for this purpose.

4.3 Compacting JavaScript

The SEVENSTAX Ajax modules are designed as other product parts for optimal product support and maintenance. This includes module and function headers, many inline comments and symbol information INSIDE the JavaScript file.

For release versions – especially in Embedded Systems - JavaScript files should require a minimum of resources. To shrink all JavaScript files, several tools can be used, after development process was finished. Please see chapter 5.2, Squeezers / Obfuscators for details.

4.4 SEVENSTAX Ajax Library

SEVENSTAX provides the JavaScript module 'ajax.js' as a Ajax library part of the product. Of course, the experienced HMTL- and JavaScript designer is able to create his own library. But this SEVENSTAX Ajax Library eases usage of Ajax-XHR-Requests, handling of JSON file and generating of dynamic HTML code with ready to use and tested code.

This is an overview of the following chapters:

- stxFct_LIB_xxx – Library Initialisation
- stxFct_ELM_xxx – Browser independent usage of Handles, Objects and DynHTML
- stxFct_HTML_xxx – Dynamic HTML Code Generation
- stxFct_XHR_xxx – XML Header Request (XHR) Support
- stxFct_DBG_xxx – Runtime DebugOut support
- stxFct_IFR_xxx – I-Frame support
- stxFct_INPUT_xxx – HMTL Input Field support
- stxFct_MATH_xxx – Mathematical support
- stxFct_STYLE_xxx – CSS Style Manipulation
- stxFct_TIME_xxx – Time Measurement support

Note: The following chapters give a short glance to the available functions. For details, please take a look into the file “ajax.js”.

4.4.1 stxFct_LIB_xxx – Library Initialisation

stxFct_LIB_Init()	Has to be called first to enable the browser abstraction layer for element handling.
stxFct_LIB_LoadScript()	Can be used to load an additional script at runtime.

4.4.2 stxFct_ELM_xxx – Browser independent usage of Handles, Objects and DynHTML

stxFct_ELM_GetAttribute()	Returns the *.Attribute* part of an HTML object.
stxFct_ELM_GetContent()	Returns the *.nodeValue* or *.innerText* of an element.
stxFct_ELM_GetHandle()	Universal function to search for a given Object (id, name, index) inside the page, returns the handle to it.
stxFct_ELM_SetContent()	Function to search for a given Object (id, name, index) inside page and to set the raw content. **Not to be used to insert HMTL-Code!** Use *stxFct_ELM_HtmlSetContent()* instead!
stxFct_ELM_HtmlSetContent()	Universal function to search for a given Object (id, name, index) inside the page and to **set** the HTML content.
stxFct_ELM_HtmlAddContent()	Universal function to search for a given Object (id, name, index) inside the page and to **add** the HTML content.
stxFct_HDL_GetParent()	Function to retrieve the handle to a parent frame (if any). Usable in Web pages with I-Frames.

4.4.3 stxFct_HTML_xxx – Dynamic HTML Code Generation

stxFct_HTML_FormOpen()	Generates HTML code to open a form with attributes,**<form..>**
stxFct_HTML_FormInputTextAdd()	Generates HTML code to open a INPUT form with attributes, **<input ...>**
stxFct_HTML_FormClose()	Generates HTML code to close a form, **</form>**
stxFct_HTML_TableAdd()	Generates HTML code to open a table with optional attributes and content, **<table..>** If content is given, closes the table, too.
stxFct_HTML_TableClose()	Generates HTML code to close a table, if not already closed inside *stxFct_HTML_TableAdd()* **</table>**.
stxFct_HTML_TableColGroupWidth()	Generates HTML code to format column group width of a table, **<colgroup width="..>**.
stxFct_HTML_TableColWidth()	Generates HTML code to format a variable number of table columns **<col width=..>**.
stxFct_HTML_TableDataAdd()	Generates HTML code to add content of a variable number of table data columns **<td..>** with optional attributes, and closes the table data, too.
stxFct_HTML_TableHeadAdd()	Generates HTML code to add a table header with optional attributes variable number of content for columns and closes it too, **<th..>**
stxFct_HTML_TableRowAdd()	Generates HTML code to add a **complete table row**, with optional attributes and variable number table data columns content, **<tr ...>** If content is given, closes the table row, too.
stxFct_HTML_TableRowClose()	Generates HTML code to close a table row, if not already closed by *stxFct_HTML_TableRowAdd(),* ***</tr>***

4.4.4 stxFct_XHR_xxx – XML Header Request (XHR) Support

stxFct_XHR_Create()	Tries to **creates** an browser specific **XHR** object, essential before sending GET/POST requests.
stxFct_XHR_GetFile()	Tries to transmit a **GET** request for a given file and registers a callback for any results.
stxFct_XHR_PostFile()	Tries to transmit a **POST** request with a given file and registers a callback for any results.
stxFct_XHR_PostTextFile()	Tries to transmit a **POST** request with a given **text block**, and registers a callback for any results.
stxFct_XHR_ValidateJson()	Tries to validate the result of XHR request to a JSON file by calling **JSON.parse()**, returns a valid JSON object on success.
stxFct_XHR_CheckResult()	Analyses the results of an XHR request, simplifies the return code

4.4.5 stxFct_DBG_xxx – Runtime DebugOut support

stxFct_DBG_Setup()	Initializes the usage of the DebugOut functions, requires a HTML ** ** id, to connect all strings to show via *stxFct_DBG_ShowText()* to that HTML area.
stxFct_DBG_ShowText()	Very simple API to show Text with Timestamp inside user specific HTML area (defined via *stxFct_DBG_Setup())*
stxFct_DBG_ToggleVisibility()	Toggles the **.visibility** attribute of the defined DebugOut area.

4.4.6 stxFct_IFR_xxx – I-Frame support

stxFct_IFR_Init()	Initializes support of I-Frame control to parent frame
stxFct_IFR_AdaptSize()	Calculates the current I-Frames size and adapts the view area of the parent frame, so that the I-Frame fits into it.

4.4.7 stxFct_INPUT_xxx – HMTL Input Field support

stxFct_INPUT_Check_Count()	Removes any character which is not a number.
stxFct_INPUT_Check_Count_phone()	Removes any character which is not a number or '+' and allows 2..21 character
stxFct_INPUT_Check_Email()	Removes all characters which do not comply to an email address string
stxFct_INPUT_Check_Server()	Removes any character which does not comply to a server domain name (URL)
stxFct_INPUT_Check_String()	Removes any special characters.
stxFct_INPUT_Check_TextBox()	Removes special characters only, which are not allowed insde text boxes.
stxFct_INPUT_Check_Time()	Removes any character which is not a number or '-'

4.4.8 stxFct_MATH_xxx – Mathematical support

stxFct_MATH_Div()	Supports Integer division, which is not available as native JavaScript.

4.4.9 stxFct_STYLE_xxx – CSS Style Manipulation

stxFct_STYLE_Visibility()	Changes the elements visibility state.

4.4.10 stxFct_TIME_xxx – Time Measurement support

stxFct_TIME_GetMilliSec()	Returns current time stamp in ms.

5 Tools

We here list some proposals for tools, which we experienced as very helpful in Ajax development process. We distinguish these kinds of tools:

- Validators – inspect and validate code and data before executing
- Squeezers – reduce and protect JavaScript code
- PlugIns – Browser Add-Ons to help debugging and analysing JavaScript code
- Webservers – rapid test of HTML and JavaScript without a target

5.1 Validators

Validators help to increase the development speed in implementation phase. Especially JavaScript is a very sensible matter, because Browsers often stop script execution on small implementation errors.

5.1.1 Jsl.exe

- Source: http://www.javascriptlint.com/
- Usage: Small command line tool, can be used like a compiler inside the IDE
- Hints: Must be configured, in order to prevent non-essential warnings/errors

5.1.2 Jslint.com

- Source: http://www.jslint.com/
- Usage: Website, needs Copy&Paste to validate JavaScript
- Hints: Must be configured, in order to prevent non-essential warnings/errors

5.1.3 Jsonlint

- Source: http://www.jsonlint.com/
- Usage: Website, needs Copy&Paste to validate JSON files
- Hints. Very helpful to reformat and analyse JSON file data

5.2 Squeezers / Obfuscators

Supporting JavaScript on embedded device often fails because of the reduced system resources. This can be improved by using **Squeezers**, which remove unnecessary JavaScript code without changing it. This improves download performance too.

If JavaScript Know-how has to be protected, **Obfuscators** are the right tools: They not only squeeze the code, they additionally change symbol and data names, therefore make it anonymous for users. But this needs very thorough system know how, because many JavaScript symbols must not be changed and execution can be deranged.

5.2.1 Jsmin.exe

- Source: http://www.crockford.com/javascript/jsmin.html

- Usage: Small command line tool to remove unnecessary JavaScript code
- Hints: very efficient in conjunction with a batch file

5.2.2 Jasob

- Source: http://www.jasob.com/
- Usage: Complete JavaScript shrink and obfuscate engine for complete projects
- Hints: with costs

5.3 Browser Plugins

Most browser to support there own JavaScript debug engine inside. Although we recommend to use the following Mozilla Add-Ons:

5.3.1 Mozilla - FireBug

- Source: https://addons.mozilla.org/de/firefox/addon/1843
- Usage: Complete JavaScript, HTML, CSS, DOM debug engine.
- Hints: A must for development Ajax Process

5.3.2 Mozilla - WebDeveloper

- Source: https://addons.mozilla.org/de/firefox/addon/60
- Usage: Some small helper to configure the browser for tests and evaluate code
- Hints: Very helpful accessory to FireBug

5.4 Test Webservers

Although SEVENSTAX Webserver is the target system, simulating the Web pages and JavaScripts on a PC Webserver have been experienced as rapid and reliable way to develop Ajax applications.

Note: Target-specific dynamic JSON files might not be supported by a PC Webserver. But this can be by-passed by using predefined JSON files inside the server directories.

5.4.1 Abyss Webserver

- Source: http://www.aprelium.com/
- Usage: Free Webserver for Web Page and JavaScript test environment
- Hints: Very easy to use and configure web server

5.4.2 Apache Webserver

- Source: http://www.apache.org/
- Usage: Free Webserver for Web Page and JavaScript test environment
- Hints: Well known standard Webserver with maximum configuration features

6 Change History

Vers.	*Date*	*by*	*Change description*
1.1	08/07/09	krz	Base version
1.2	11/12/09	krz	Complete review based on API update
1.3	17/02/10	jma	Textual review
1.4	19/02/10	krz	Added UTF-8 module Added program flow for HTTP-POST and GET Requests

The information furnished in this document is believed to be accurate and reliable. However, no responsibility is assumed by sevenstax for its use, nor for any infringements of patents or other rights of third parties resulting from its use. No license is granted under any patents or patent rights of sevenstax.

This document is an intellectual property of sevenstax GmbH. Unauthorized copying and distribution is prohibited.

Copyright (c) 2010 by SEVENSTAX GmbH